COURS SYSTÈME D'EXPLOITATION

Les processus

Plan

- **Introduction**
- **le cycle de vie d’un processus**
- **Commutation des processus**
- **Ordonnancement des processus**
- **Communication interprocessus**
- **Synchronisation de processus**

Processus : Introduction

Un processus est un programme en cours d'exécution:

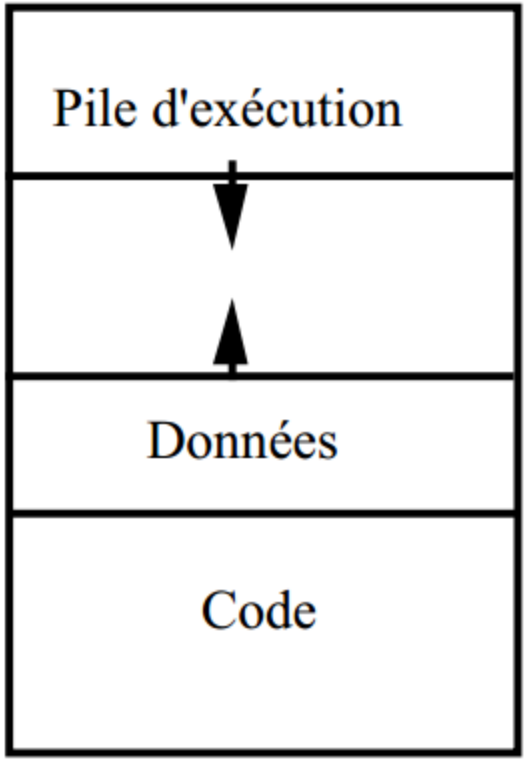
- C’est un concept dynamique qui représente l’exécution d'une tâche faisant partie d'une application ou un programme système quelconque.
- L’exécution d’un processus est une alternance de calculs effectués par le processeur et de requêtes d’Entrée/Sortie effectuées par les périphériques.

Le système a pour fonction de les créer, les gérer, les synchroniser, ainsi que de leur permettre de communiquer entre eux.

Processus : Introduction

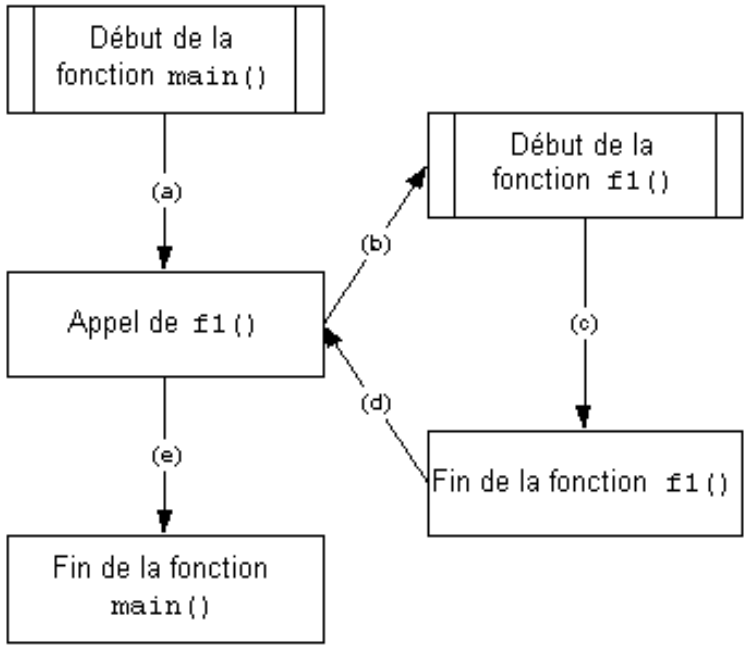
À chaque processus il est réservé un espace de travail (un espace d'adressage) en mémoire formé de trois segments essentielles :

- **Segment de code** : pour stocker les instructions.
- **Segment de données** : pour stocker les données.
- **Pile d'exécution**



Processus : Introduction

Pile d'exécution : Est une structure de donnée qui sert principalement à garder l'endroit en mémoire où chaque sous-programme actif dans le programme en cours d'exécution, doit retourner à la fin son exécution.



→Une pile d'adresses de retour après avoir appeler des sous-programmes.

Plan

- **Introduction**
- **Cycle de vie d'un processus**
  - **Création et terminaison de processus**
  - **États et Transitions d'état des processus**
- **Commutation des processus**
- **Ordonnancement des processus**
- **Communication interprocessus**
- **Synchronisation**

Création et terminaison des processus

- Le système d'exploitation dispose d'un ensemble d'appels système permettant,
  - la création, la destruction, la communication et la synchronisation de processus.
- Au lancement du système, il n'existe qu'un seul processus, qui est l'ancêtre de tous les autres.

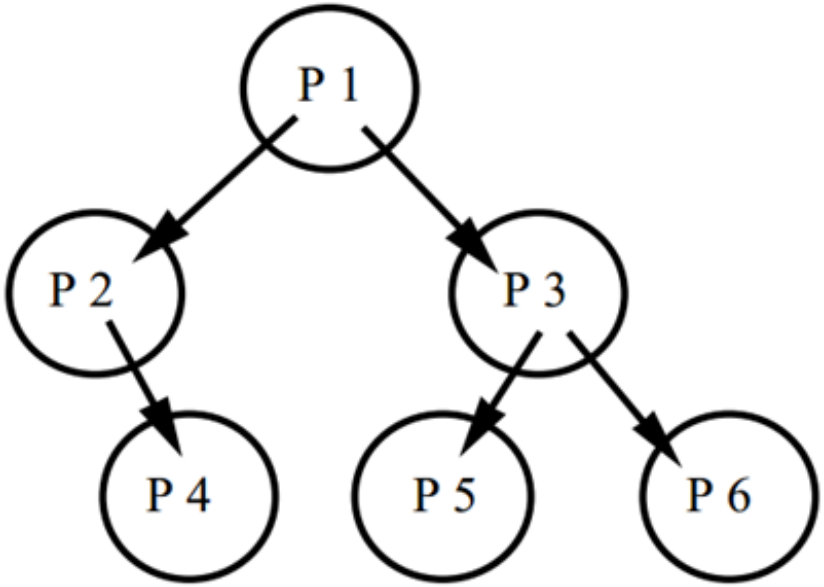
Création d'un nouveau processus

Evénements conduisant à la création d'un nouveau processus :

- Initialisation du système.
- Requête utilisateur sollicitant la création d'un nouveau processus.
- Exécution d'un appel système de création de processus par un autre processus.
  - Un processus peut, durant son cycle de vie, créer un ou plusieurs processus fils qui, à leur tour, peuvent créer des processus fils sous une forme de structure arborescente.
  - Le processus créateur est appelé processus père.

Création d'un nouveau processus

Les processus peuvent se structurer sous la forme d'une arborescence.



Création d'un nouveau processus: Processus père-fils

- Un processus fils peut **partager** avec son père, certaines ressources comme la mémoire ou les fichiers.
- Le processus père peut **contrôler l'usage des ressources** partagées et peut avoir une **certaine autorité** sur ses processus fils.
  - Également, il peut les suspendre ou les détruire.
- Il peut également se mettre en attente de la fin de l'exécution de ses fils.

Création d'un nouveau processus: Processus père-fils

Exécution des processus père – fils :

- Exécution séquentielle : dans certains SE, l'exécution du processus père est suspendue le temps que les processus fils s'exécutent.
- Exécution asynchrone : dans d'autres systèmes le processus père continue à s'exécuter en concurrence avec ses fils.

Terminaison des processus

- Un processus se termine par une demande d'arrêt volontaire ou par un arrêt provoqué par un autre processus.
- Lorsqu'un processus se termine, toutes les ressources systèmes qui lui ont été allouées sont libérées par le système d'exploitation.

Plan

- **Introduction**
- **Cycle de vie d'un processus**
  - **Création et terminaison de processus**
  - **États et Transitions d'état des processus**
- **Commutation des processus**
- **Ordonnancement des processus**
- **Communication interprocessus**
- **Synchronisation de processus**

Les états d'un processus

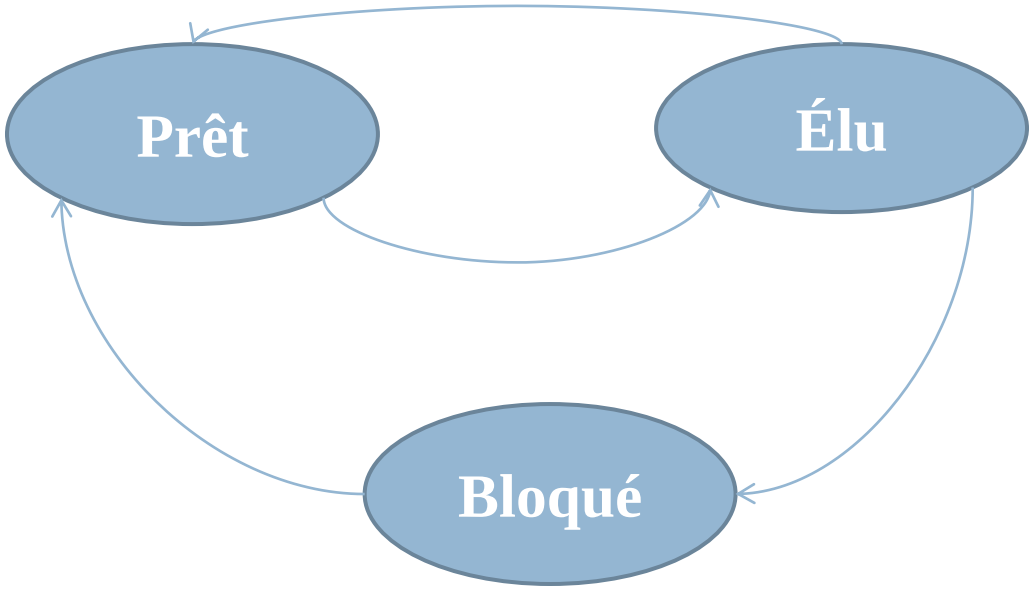
Dans un système multitâches le temps du processeur doit être partagé entre les processus.

→Donc un processus n'est pas continuellement en exécution, il a trois états principaux :

- Élu : Le processus est en cours d'exécution.
- Prêt : Le processus est actif en mémoire centrale et prêt d'être exécuté.
  - Après la création d'un processus il est à l'état prêt.
- Bloqué : Le processus est en attente d'un événement extérieur.

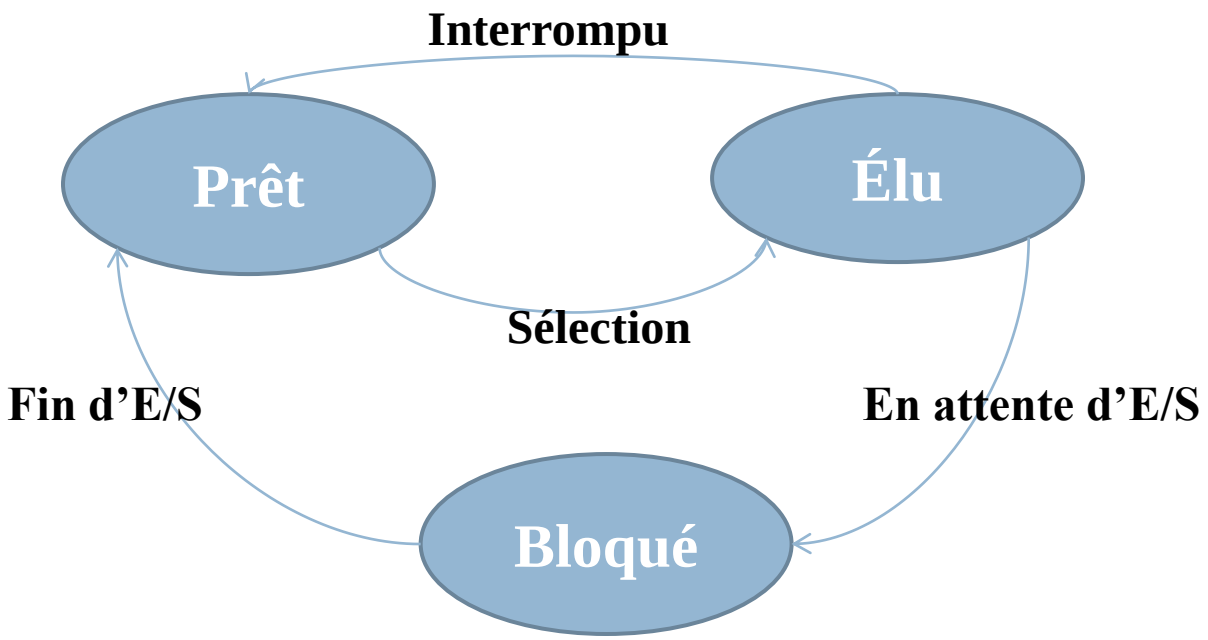
Les états d'un processus

Le diagramme simplifié des états d'un processus est donc :



Transitions d’état des processus

Il y a quatre transitions possible :



Transitions d'état des processus

- Transition 1 : Le processus (**est élu**) passe de l'état Prêt à l'état Elu,
  - lorsque le processeur entame son exécution
- Transition 2 : Le processus (**est interrompu**) passe de l'état Elu à l'état Prêt,
  - la tranche de temps qui lui est accordé est achevée,
  - un processus de plus haute priorité réquisitionne le processeur.
  - …

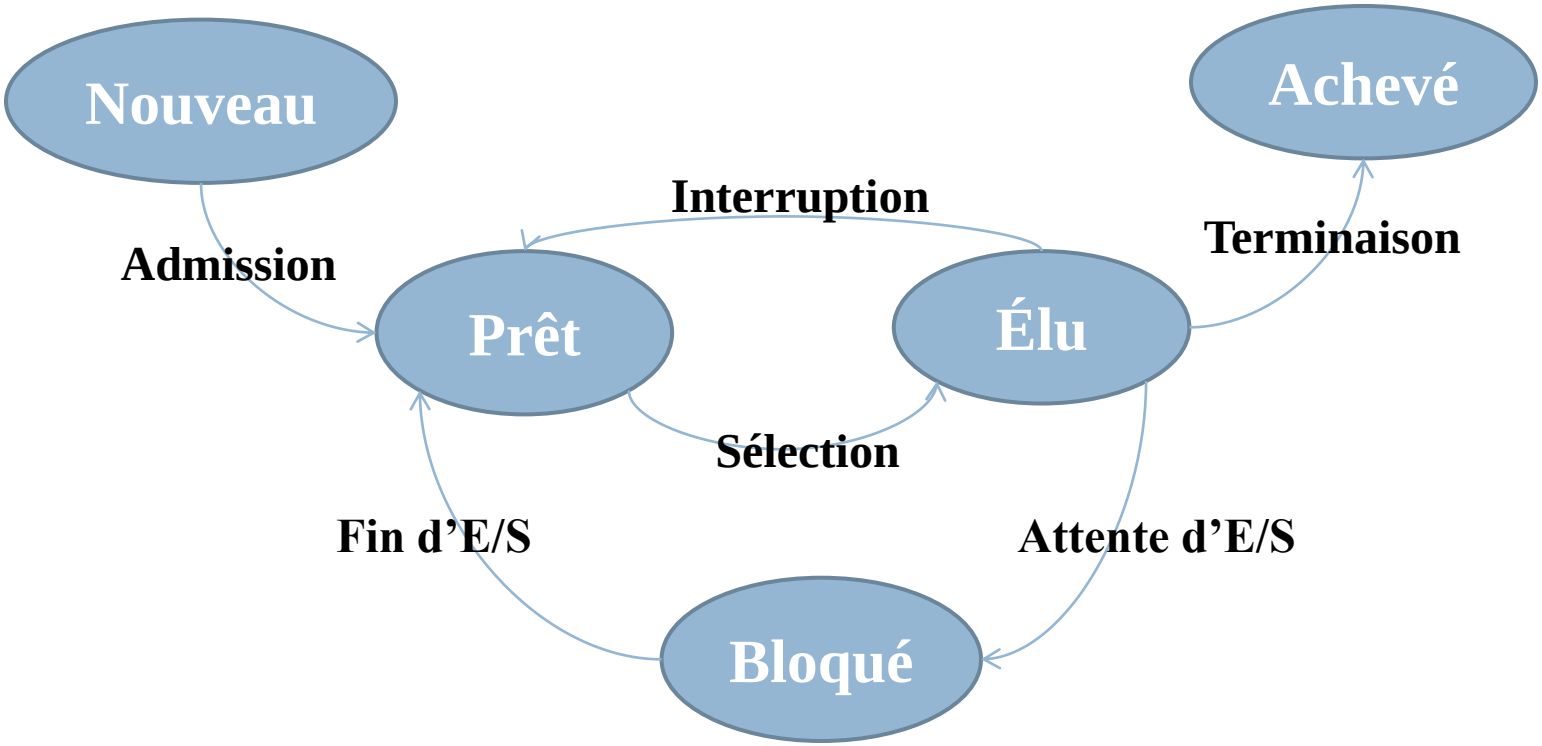
Transitions d'état des processus

- Transition 3 : Le processus (**est bloqué**) passe de l'état Elu à l'état Bloqué
  - le processus en attente d'une opération d'E/S

- Transition 4 : Le processus (**est débloqué**) passe de l'état Bloqué à l'état Prêt
  - lorsque l'événement qui a bloqué l'exécution d'un processus survient→il redevient prêt.

Transitions d'état des processus

Le modèle à trois états est complété par deux états supplémentaires :



Transitions d’état des processus

- **Nouveau :** indique que le processus vient d'être créé, en attendant d'être admis par l’ordonnanceur en tant que processus Prêt.
- **Achevé :** indique la terminaison de l’exécution d’un processus.
  - Dans cet état le processus est désormais inactif car il a achevé sa tâche.
  - Il sera détruit prochainement par le système d'exploitation pour libérer de la place en mémoire.
  - Il est parfois conservé pendant un temps à l'état terminé en attendant qu'une entrée/sortie s'achève ou que les données de ce processus soient exploitées par un autre.

Plan

- **Processus : Définition**
- **le cycle de vie d'un processus**
- **Commutation des processus**
  - **Pseudo-parallélisme**
  - **Sauvegarde d'informations du processus PCB**
  - **Changement de contexte**
- **Ordonnancement des processus**
- **Communication interprocessus**
- **Synchronisation de processus**

Pseudo-parallélisme

- Un système d'exploitation doit en général traiter plusieurs processus en même temps.
- Un processeur ayant un seul cœur est capable d'exécuter une seule tache à la fois, donc il résout ce problème grâce au pseudo-parallélisme.
  - Il se base sur la commutation des processus.
  - Il bascule rapidement entre les processus ce qui donne l'illusion de la simultanéité d'exécution.

Sauvegarde d'informations du processus

- En multiprogrammation, un processus s'exécute d'une façon sporadique.
- À chaque fois le processeur reprend l'exécution d'un processus il doit la reprendre du même point ou il l'a laissé.

→Besoin de **sauvegarder des informations** sur chaque processus et son état dans des structures de données (**PCB**), pour pouvoir les interrompre et les relancer selon ce que décide l'ordonnanceur.

Sauvegarde d'informations du processus : PCB

- **Le Bloc de Contrôle de Processus :** est une table contenant des informations sur la situation actuelle des processus créés.
- Son contenu se diffère selon les systèmes d'exploitation, généralement un PCB contient :
  - L'identifiant du processus (PID), l'identifiant du processus parent (PPID) et l'identifiant de l'utilisateur du processus (UID)
  - L'état du processus : élu, bloqué, prêt
  - Le compteur ordinal du processus
  - L'espace d'adressage du processus : l'emplacement mémoire du code, des données et de la pile d'exécution
  - Des pointeurs vers les ressources utilisées/ouverts par le processus (fichiers, E/S, ...)
  - D'autres informations telles que le temps processeur accumulé par le processus, etc.

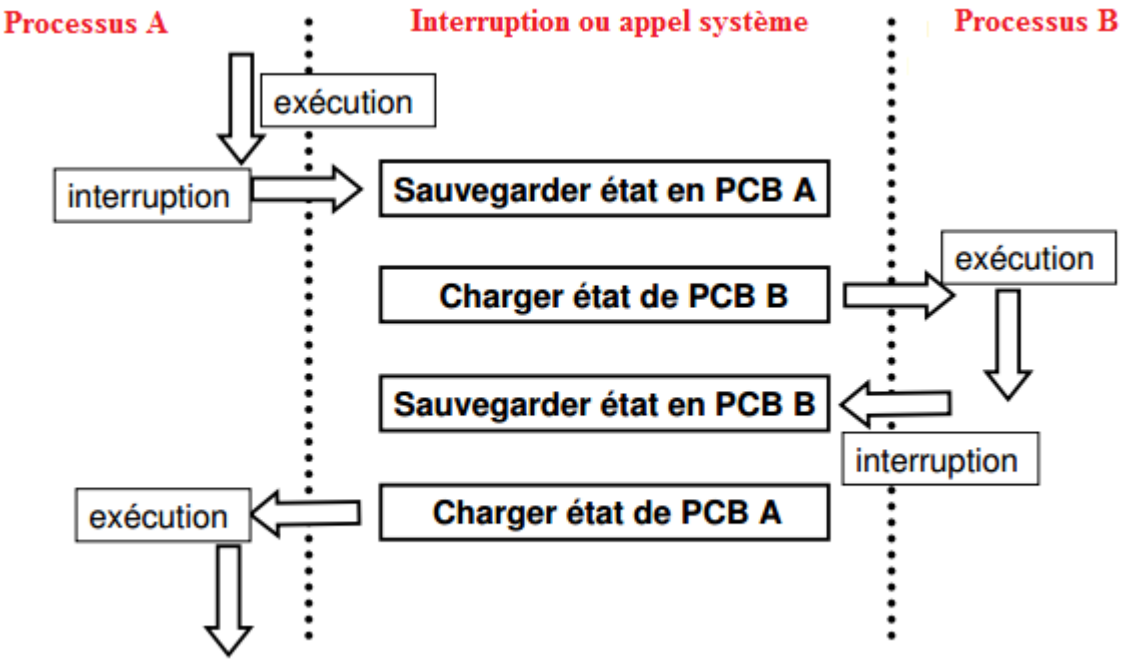
Changement de contexte

A chaque fois il y a une suspension de l'exécution d'un processus afin de donner la main au processeur pour exécuter un autre processus B, il faut :

- Mettre à jour le PCB de A
- Sauvegarder le PCB de A
- Reprendre le PCB de B
- Remettre les registres du processeur avec les valeurs décrites dans le PCB du processus B

Changement de contexte

Le schéma suivant illustre le changement de contexte entre deux processus



Plan

- **Processus : Définition**
- **le cycle de vie d'un processus**
- **Commutation des processus**
- **Ordonnancement des processus**
  - **Définition / Objectif**
  - **Les types d'ordonnanceur**
  - **Les algorithmes d'ordonnancement**
- **Communication interprocessus**
- **Synchronisation de processus**

Ordonnancement des processus : Définitions

- La concurrence des processus est basée sur le partage du temps du processeur, de la mémoire et des entrées/sorties.
- L'organisation de la concurrence est devenue possible grâce au concept d'ordonnanceur.
- Un ordonnanceur : c'est celui qui décide quel processus doit passer à quel état et au quel moment.

Ordonnancement des processus : Objectifs

Les objectifs d'un ordonnanceur sont :

- Maximiser l'utilisation du processeur
- Être équitable entre les différents processus
- Assurer certaines priorités
- Minimiser le temps de réponse
  - Le temps entre une demande et la réponse du système (la vitesse de réaction aux interventions extérieures)
- Minimiser le temps d'attente
  - Le temps d'attente passé en file Prêt
- Etc.

Plan

- **Processus : Définition**
- **le cycle de vie d'un processus**
- **Commutation des processus**
- **Ordonnancement des processus**
  - **Définition / Objectif**
  - **Les types d'ordonnanceur**
  - **Les algorithmes d'ordonnancement**
- **Communication interprocessus**
- **Synchronisation de processus**

Types d'ordonnanceurs

Il est possible de distinguer trois types d'ordonnanceurs :

- **À long terme** : sa vocation est de sélectionner les programmes à admettre pour leur exécution
  - Les programmes admis deviennent des processus à l'état **Prêt**.
  - L'admission implique l'embarquement du processus dans la mémoire centrale
  - Les processus doivent être assez nombreux pour que le processeur soit inactif le plus rarement possible sans saturer la mémoire principale du système.

Remarques :

Une fois le processus est admis dans le système, il n'en sort que lorsqu'il est terminé ou détruit.

Types d’ordonnanceurs

- **À moyen terme** : il sélectionne parmi les processus déjà admis (à l’état Prêt) qui embarque/débarque de la mémoire centrale, en fonction de :
  - l’espace mémoire
  - et des requête d’E/S des périphériques

Remarques :

- Les permutations effectués ne peuvent pas être trop fréquentes pour ne pas gaspiller la bande passante des disques.
- L'ordonnancement à moyen terme est assuré par l'**ordonnanceur de mémoire** aussi appelé *swapper*

Types d'ordonnanceurs

- À **court terme** : s'occupe de la gestion de la file des processus en état **Prêt**.
  - Il choisit à quel processus sera alloué le processeur et pour combien de temps.

Remarques :

- L'ordonnanceur est activé par un événement : interruption du temporisateur, interruption d'un périphérique, appel système ou signal.
- L'ordonnanceur à court terme peut implanter un ordonnancement non-préemptif ou préemptif.

Plan

- **Processus : Définition**
- **le cycle de vie d'un processus**
- **Commutation des processus**
- **Ordonnancement des processus**
  - **Définition / Objectif**
  - **Les types d'ordonnanceur**
  - **Algorithmes d'ordonnancement**
- **Communication interprocessus**
- **Synchronisation de processus**

Algorithmes d'ordonnancement

On distingue deux politiques d'ordonnancement :

- Ordonnanceur non préemptif (sans réquisition)
- Ordonnanceur préemptif (avec réquisition)

Ordonnanceurs non préemptifs

- Dans ce genre d'ordonnanceur le processeur est alloué au processus jusqu'à ce qu'il soit terminé ou bloqué.
  - Il n'y a pas de réquisition.
- Les ordonnanceurs non préemptifs ne sont pas intéressants pour les systèmes multi-utilisateurs car le temps de réponse n'est pas toujours acceptable.

Ordonnanceurs préemptifs

- Un **ordonnanceur préemptif**, ou avec réquisition, permet de s'assurer qu'aucun processus ne s'exécute pendant trop de temps.
- Dans la plus part des cas le processeur passe d'un processus à un autre en exécutant chaque processus pendant quelques dizaines ou centaines de millisecondes.

PAPS = FCFS

Premier Arrivé est le Premier servi

L'algorithme PAPS **appelé encore** First-Come First Served (FCFS).

- Dans cet algorithme les processus sont enfilés dans une file de processus prêts,
- à chaque interruption le système décide d'allouer le processeur au processus en tête de la file,
- jusqu'à ce qu'il termine son exécution ou se bloque.

Remarques :

- C'est un algorithme non préemptif
- Il est facile à écrire et à comprendre, mais peu efficace ..

SJF

Plus Court d'abord (SJF)

L'algorithme SJF (Shortest Job First) :

- Dans cet algorithme le processus ayant le temps d'exécution le plus court est exécuté le premier.

Remarques :

- Il est optimal en point de vue du *temps d'attente*
- Il est non préemptif (sans réquisition)
- Il y a une affectation implicite de priorité : préférence aux travaux plus courts

Plus Court d'abord (SJF) : Critiques

- Les processus ayant un temps d'exécution plus longs peuvent souffrir de la *famine*
- Un processus long peut monopoliser le processeur s'il est le premier à s'exécuter et il ne fait pas d'E/S
  - La préemption est nécessaire pour les environnements à multitâches.

Plus Court d'abord (SJF) : cas particulier

Si un processus, qui dure moins de temps que le *restant* du processus en cours d'exécution, vient de d'arriver:

- Dans un environnement **sans préemption** : On permet au processus courant de terminer son exécution.
- Dans un environnement **avec préemption** : le processeur est alloué au nouveau processus.

→**SRTF**: Shortest Remaining-Time First

SRTF

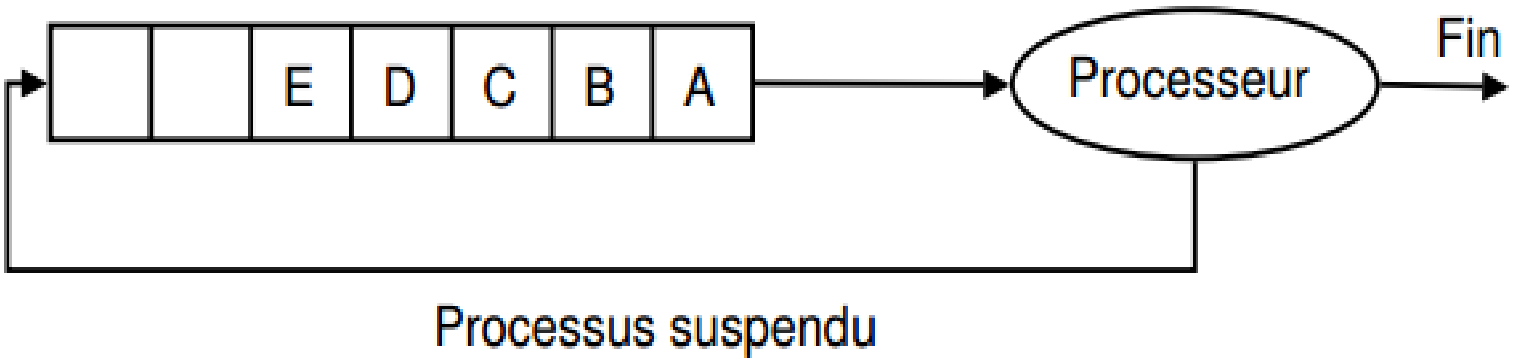
Le Plus Court Temps Restant (SRTF)

- L'algorithme **Shortest Remaining Time First** (SRTF) est la version préemptive de l'algorithme SJF.
- L'arrivée d'un nouveau processus interrompe l'exécution du processus en cours,
- l'ordonnanceur compare la valeur du temps d'exécution espérée pour ce nouveau processus contre la valeur du processus actuellement en exécution,
- si le temps du nouveau processus est plus petit, il rentre en exécution immédiatement.

RR=tourniquet

Tourniquet ou Round-Robin (RR)

- C'est l'un des algorithmes les plus utilisés en pratique
- À chaque processus il est alloué un quantum (*tranche*) de temps (par exemple 10-100 millisecs) pour s'exécuter
- Une fois le quantum du temps est expiré, l'exécution du processus est interrompu et le processeur est alloué à un autre processus



Tourniquet ou Round-Robin (RR)

Remarques :

- Le processus interrompu redevient prêt en queue de la file.
- Si le processus se bloque ou se termine avant la fin de son quantum, le processeur est immédiatement alloué à un autre processus (celui en tête de file).
- Les processus qui arrivent sont insérés en queue de file.
- Le seul paramètre important à régler, pour le tourniquet, est la durée du quantum.
- Il est préemptif.

Tourniquet ou Round-Robin (RR)

Choix de la valeur du quantum du temps :

- Un quantum trop petit provoque trop de commutations de processus et abaisse l'efficacité du processeur.
- Un quantum trop élevé augmente le temps de réponse des courtes commandes en mode interactif.

Priorité

L’ordonnanceur à priorité

- Cet algorithme attribue à chaque processus une priorité (par .ex. un nombre entier) en se basant sur sa nature (les processus du système sont plus urgents est donc sont plus prioritaires).
- Souvent les petits chiffres dénotent des hautes priorités (0 la plus haute).
- Le processeur est alloué au processus prêt avec la plus haute priorité.

L’ordonnanceur à priorité

Remarques :

- Il y a une file de processus en état Prêt pour chaque priorité.
- En général, les processus de même priorité (de la même file) sont ordonnancés selon l’algorithme du tourniquet.
- Politique préemptive : si un nouveau processus plus prioritaire arrive, le processeur lui est alloué directement.
- Politique non préemptive : le processus en cours d’exécution termine son cycle.

L’ordonnanceur à priorité

Problématique : les processus moins prioritaires n’arrivent jamais à s’exécuter ce qui peut causer le problème de la famine (vieillissement des processus moins prioritaires)

Solution : priorité dynamique

modifier la priorité d’un processus en fonction de son âge et de son historique d’exécution (le processus change de file d’attente).

L’ordonnanceur à priorité

Evolution des priorités :

- Pour empêcher les processus de priorité élevée de s’exécuter indéfiniment, l’ordonnanceur diminue régulièrement la priorité du processus en cours d’exécution.
- La priorité du processus en cours est comparée régulièrement à celle du processus prêt le plus prioritaire (en tête de file).
- Lorsqu’elle devient inférieure, la commutation a lieu.
- Dans ce cas, le processus suspendu est inséré en queue de la file correspondante à sa nouvelle priorité.

Files multiples à retour (quantum variable)

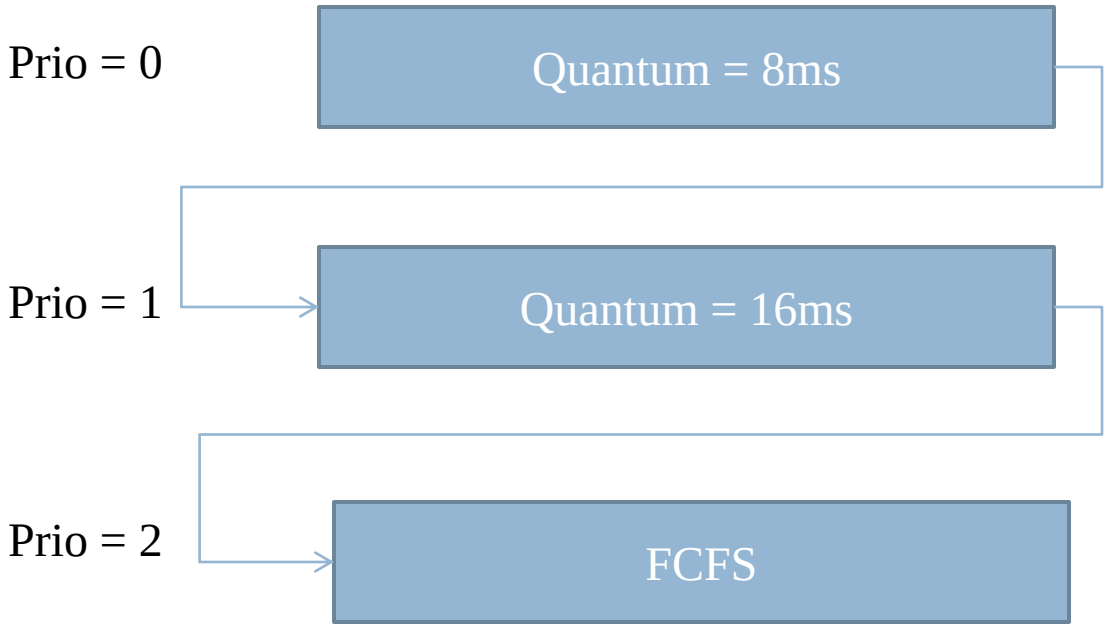
Files multiples à retour (quantum variable)

Certains processus se sont caractérisé par une consommation élevée de temps d'exécution, et puisque les commutations des processus et couteuse en temps, alors il est préférable de leur allouer un plus grand quantum.

- Lorsqu'un processus passe à l'état Prêt :
  - Pour la première fois, le processeur lui est alloué pendant un quantum.
  - Pour la seconde fois, le processeur lui est alloué pendant 2 quantum...
  - Pour la nième fois, le processeur lui est alloué pendant 2^n quantum.
  - Ou pour certain moment le processus peut passer à une file dont l'algorithme d'ordonnancement est FCFS.

Files multiples à retour (quantum variable)

Exemple :



Files à plusieurs niveaux (multiples)

Files à plusieurs niveaux

- Dans cette stratégie la file des processus en état Prêt est séparée en plusieurs files par ex. :
  - processus système
  - processus interactif
  - …
- Et chaque file à sa propre stratégie d'ordonnancement par ex. :
  - FCFS pour la première file de processus système
  - Tourniquet pour la deuxième file

- Comment ordonnancer entre files? Deux propositions :
  - Priorité fixe à chaque file (famine possible)
  - Chaque file reçoit un certain pourcentage de temps du processeur, par ex. :
    - 80% pour arrière-plan
    - 20% pour premier plan

Files à plusieurs niveaux

Exemple :

Plus haute priorité

Processus système

Processus Interactifs

Processus batch

Processus d'autres utilisateurs

Plus basse priorité

Processus

Plan

- Communication interprocessus
  - Introduction
  - Les signaux
  - Les tubes de communication
- Synchronisation de processus
  - Introduction
  - Sections critiques
  - Comment assurer l'exclusion mutuelle ?

Plan

- **Introduction**
- **le cycle de vie d'un processus**
- **Commutation des processus**
- **Ordonnancement des processus**
- **Communication interprocessus**
  - Introduction
  - Les signaux
  - Les tubes de communication
- **Synchronisation de processus**

Introduction

- Dans certains cas nous avons besoin de coopérer plusieurs processus afin de résoudre une problématique donnée.

→**Communication des processus impliqués** par :

- Mémoires et fichiers partagés
- **les signaux**
- **Tubes de communication**
- Sockets

Les signaux : définition

Les **signaux** (appelés aussi interruptions virtuelles) sont utilisés par le système d'exploitation pour :

- Aviser les processus de l'occurrence d'un événement important.
- Etablir une communication minimale entre processus.
- Permettre une communication avec le monde extérieur.
- Effectuer la gestion des erreurs.

Les signaux

A la réception d'un signal le processus destinataire peut :

- **Ignorer le signal**
  - Certains signaux ne peuvent être ignorés.
- **Appeler une routine de traitement fournie par le noyau**
  - Tuer le processus.
  - Dans certains cas, la création d'un fichier core (contient le contexte du processus avant de recevoir le signal).
- **Appeler une procédure spécifique créée par le programmeur**
  - Certains signaux ne permettent pas ce type d'action.

Les tubes de communication : définition

Les **tubes de communication** ou *pipes* (structures de données /outil) permettent à deux ou plusieurs processus d'échanger des informations.

On distingue deux types :

- Tube sans nom (unamed pipe)
- Tube nommé (named pipe).

Tube sans nom (unamed pipe)

Un **tube sans nom est** :

- Une liaison unidirectionnelle de communication.
- Sa taille est limitée.
- Les processus impliqués dans la communication s'exécutent en concurrence.

Un tube sans nom est, en général, utilisé pour la communication entre **un processus père et ses processus fils**.

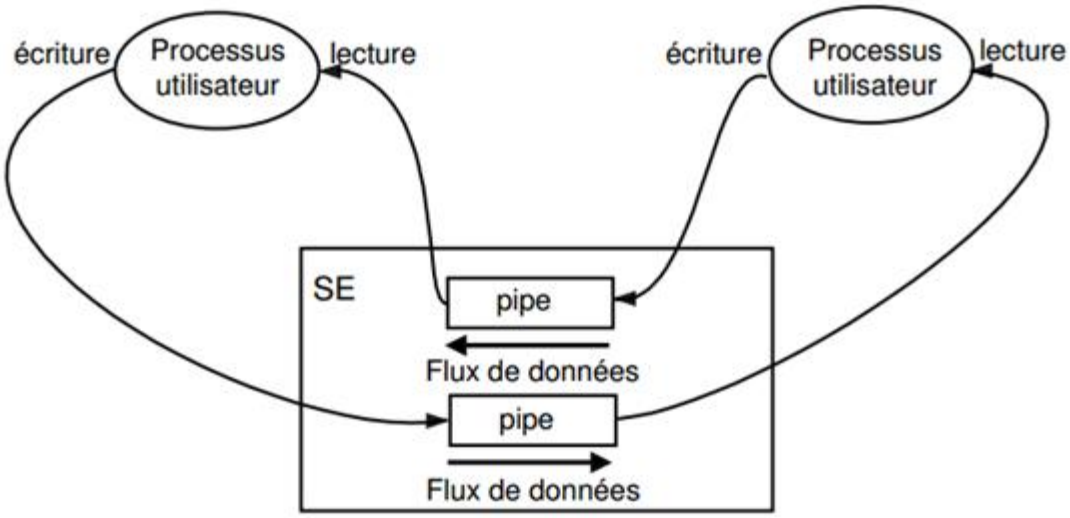
Tube sans nom (unamed pipe)

Procédure de communication :

- Les sorties du premier processus sont stockées sur le tube de communication.
- Une fois le tube devient plein, le premier processus (**écrivain**) est suspendu jusqu'à ce qu'il y ait libération de l'espace nécessaire pour stocker encore de l'information.
- De façon similaire, lorsque le tube devient vide, le second processus (**lecteur**) est suspendu jusqu'à ce qu'il y ait au moins une ligne d'information sur le tube.

Tube sans nom : Communication bidirectionnelle

La communication bidirectionnelle entre processus est possible en utilisant deux tubes : un pour chaque sens de communication



Tubes de communication nommés

- Linux supporte un autre type de tubes de communication, beaucoup plus performant. Ils s'agit des **tubes nommés** (named pipes).
- Ils offrent plus d'avantages par rapport aux **tubes sans nom** :
  - Ils ont chacun un nom qui existe dans le système de fichiers (une entrée dans la table de fichiers).
  - Ils sont considérés comme des fichiers spéciaux.
  - Ils peuvent être utilisés par des processus indépendants, à condition qu'ils s'exécutent sur une même machine.
  - Ils sont de capacité plus grande.

Tubes de communication nommés

Procédure de communication :

- Une fois le tube créé, il peut être utilisé pour réaliser la communication entre deux processus.
- Chacun des deux processus ouvre le tube, l'un en mode écriture et l'autre en mode lecture.

Plan

- **Introduction**
- **le cycle de vie d'un processus**
- **Commutation des processus**
- **Ordonnancement des processus**
- **Communication interprocessus**
- **Synchronisation de processus**
  - Introduction
  - Exclusion mutuelle : définition
  - Comment assurer l'exclusion mutuelle ?

Synchronisation : introduction

- Dans un système d'exploitation multiprogrammé, plusieurs processus s'exécutent en parallèle et partagent des ressources (mémoires, imprimantes, etc.).
- Le partage d'objets sans précaution particulière peut conduire à des résultats imprévisibles.

  →**Synchronisation entre les processus**.

Exclusion mutuelle

Dans un environnement à accès concurrent, le gestionnaire des ressources communes doit empêcher tous autres processus d'accéder à un objet partagé si cet objet est en cours d'utilisation par un processus.

→L'accès aux objets critiques doit se faire **en exclusion mutuelle**

Exclusion mutuelle : Objets et sections critiques

L'exclusion mutuelle impose deux concepts :

- **Objet critique :** Objet qui ne peut être accédé simultanément.
- **Section critique :** suites d'instructions qui opèrent sur un ou plusieurs objets critiques et qui ne doivent pas être exécutées simultanément par des processus différents.

Exclusion mutuelle

- Le problème de conflit d'accès serait résolu, si on pouvait assurer que deux processus ne soient jamais en section critique en même temps au moyen de l'**exclusion mutuelle**.
- En général, les processus qui exécutent des sections critiques sont structurés comme suit :
  - Section non critique.
  - Demande d'entrée en section critique.
  - Section critique.
  - Demande de sortie de la section critique.
  - Section non critique.

Exclusion mutuelle

Quatre conditions sont nécessaires pour réaliser correctement une exclusion mutuelle :

1. Deux processus ne peuvent être en même temps en section critique.
2. Aucune hypothèse ne doit être faite sur les vitesses relatives des processus et sur le nombre de processeurs.
3. Aucun processus suspendu en dehors d'une section critique ne doit bloquer les autres processus.
4. Aucun processus ne doit attendre trop longtemps avant d'entrer en section critique.

Comment assurer l'exclusion mutuelle?

- Masquage des interruptions
- Exclusion mutuelle par attente active
  - Attente active et verrouillage
  - Attente active avec alternance
  - Solution de Peterson
- Primitive *sleep() / wakeup()*
- Les sémaphores

Masquage des interruptions

Dans cette méthode un processus:

- Avant d'entrer dans une section critique, il masque les interruptions.
- Il ne peut être alors suspendu durant l'exécution de la section critique.
- Il les restaure à la fin de la section critique.

Problèmes : si le processus, pour une raison ou pour une autre, ne restaure pas les interruptions à la sortie de la section critique, ce serait la fin du système.

Attente active et verrouillage

Consiste à utiliser une variable partagée ***verrou***, unique, initialisée à 0.

- Pour entrer en section critique, un processus doit tester la valeur du verrou.
  - Si verrou = 0, le processus met le verrou à 1 puis exécute sa section critique.
  - A la fin de la section critique , il remet le verrou à 0.
  - Sinon, il attend (attente active) que le verrou devienne égal à 0.

Attente active et verrouillage : Problèmes

(1) Cette méthode n'assure pas l'exclusion mutuelle :

- Supposons qu'un processus est suspendu juste après avoir lu la valeur du verrou qui est égal à 0.
- Ensuite, un autre processus est élu. Ce dernier teste le verrou qui est toujours égal à 0, met le verrou à 1 et entre dans sa section critique.
- Ce processus est suspendu avant de quitter la section critique.
- Le premier processus est alors réactivé, il entre dans sa section critique et met le verrou à 1.
- Les deux processus sont en même temps en section critique.

(2) L'attente active surcharge le processeur

Attente active avec alternance

Consiste à utiliser une variable ***tour*** qui mémorise le tour du processus qui doit entrer en section critique.

tour est initialisée à 0.

- Pour entrer en section critique, un processus doit tester la valeur du tour.
  - Si c'est son tour (ex : tour = 0 pour P_0), le processus exécute sa section critique.
  - A la fin de la section critique , il donne le tour au processus suivant.
  - Sinon, il attend (attente active) que le tour soit à lui.

Attente active avec alternance : Problèmes

(1) Un processus peut être bloqué par un processus qui n'est pas en section critique.

- P0 lit la valeur de tour qui vaut 0 et entre dans sa section critique. Il est suspendu et P1 est exécuté.
- P1 teste la valeur de tour qui est toujours égale à 0. Il entre donc dans une boucle en attendant que tour prenne la valeur 1. Il est suspendu et P0 est élu de nouveau.
- P0 quitte sa section critique, met tour à 1 et entame sa section non critique. Il est suspendu et P1 est exécuté.
- P1 exécute rapidement sa section critique, tour = 0 et sa section non critique. Il teste tour qui vaut 0. Il attend que tour prenne la valeur 1.

(2)Attente active consomme du temps CPU.

Solution de Peterson

La solution de Peterson se base sur deux fonctions *entrer_region et quitter_region*.

- Chaque processus doit, avant d'entrer dans sa section critique appeler la fonction *entrer_region* en lui fournissant en paramètre son numéro de processus.
- A la fin de la section critique, il doit appeler *quitter_region* pour indiquer qu'il quitte sa section critique et pour autoriser l'accès aux autres processus.

Les primitives SLEEP et WAKEUP

- **SLEEP** est un appel système qui suspend l'appelant en attendant qu'un autre le réveille.
- **WAKEUP**(process) : est un appel système qui réveille le processus process.
- Un processus A qui veut entrer dans sa section critique se met en mode sleep si un autre processus B est déjà dans sa section critique.
- Le processus A sera réveillé par le processus B, lorsqu'il quitte la section critique.

Les primitives SLEEP et WAKEUP : Problème

Si le signal émis par WAKEUP() arrive avant que le destinataire ne soit endormi (avant la primitive SLEEP()), le processus peut dormir pour toujours.

Sémaphores

- Pour contrôler les accès à un objet partagé, E. W. Dijkstra a proposé un nouveau type de **variables** appelées ***sémaphores***.
- Un sémaphore est un compteur entier qui désigne le nombre d'autorisations d'accès disponibles à une ressource partagée.
- Chaque sémaphore a un nom et une valeur initiale.
- Les sémaphores sont manipulés au moyen des opérations :
  - P (désigné aussi par down)
  - V (désigné aussi par up).

Sémaphores

- Les deux opérations *P* et *V* sont atomiques
- L'opération P(S) décrémente la valeur du sémaphore *S* si cette dernière est supérieure à 0, sinon le processus appelant est mis en attente.
- L'opération V(S) incrémente la valeur du sémaphore S.

Sémaphores

- Pour éviter l'attente, un sémaphore peut avoir une file de processus associée (généralement une file du type FIFO).
  - Si un processus exécute l'opération P sur un sémaphore qui a la valeur zéro, le processus est ajouté à la file du sémaphore.
  - Quand un autre processus incrémente le sémaphore en exécutant l'opération V, et qu'il y a des processus dans la file, l'un d'eux est retiré de la file et reprend la suite de son exécution.